Accelerating Inference on Binary Neural Networks with Digital RRAM Processing

David Atienza, Pierre-Emmanuel Gaillardon, João Vieira, Edouard Giacomin, Yasir Qureshi, Marina Zapater, Xifan Tang, Shahar Kvatinsky

▶ **To cite this version:**

David Atienza, Pierre-Emmanuel Gaillardon, João Vieira, Edouard Giacomin, Yasir Qureshi, et al.. Accelerating Inference on Binary Neural Networks with Digital RRAM Processing. 27th IFIP/IEEE International Conference on Very Large Scale Integration - System on a Chip (VLSI-SoC), Oct 2019, Cusco, Peru. pp.257-278, 10.1007/978-3-030-53273-4_12 . hal-03476606

HAL Id: hal-03476606
https://inria.hal.science/hal-03476606

Submitted on 13 Dec 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Accelerating Inference on Binary Neural Networks with Digital RRAM Processing

João Vieira[1,**], Edouard Giacomin[2], Yasir Qureshi[3], Marina Zapater[3], Xifan Tang[2], Shahar Kvatinsky[4], David Atienza[3], and Pierre-Emmanuel Gaillardon[2]

[1] INESC-ID, Instituto Superior Técnico, University of Lisboa, Portugal, joaomiguelvieira@tecnico.ulisboa.pt
[2] LNIS, University of Utah, USA
[3] ESL, Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland
[4] Andrew and Erna Viterbi Faculty of Electrical Engineering, Technion, Israel Institute of Technology, Israel

Abstract. The need for efficient *Convolutional Neural Networks* (CNNs) targeting embedded systems led to the popularization of *Binary Neural Networks* (BNNs), which significantly reduce execution time and memory requirements by representing the operands using only one bit. Also, due to 90% of the operations executed by CNNs and BNNs being convolutions, a quest for custom accelerators to optimize the convolution operation and reduce data movements has started, in which *Resistive Random-Access Memory* (RRAM)-based accelerators have proven to be of interest. This work presents a custom *Binary Dot Product Engine* (BDPE) for BNNs that exploits the low-level compute capabilities enabled RRAMs. This new engine allows accelerating the execution of the inference phase of BNNs by locally storing the most used kernels and performing the binary convolutions using RRAM devices and optimized custom circuitry. Results show that the novel BDPE improves performance by 11.3%, energy efficiency by 7.4% and reduces the number of memory accesses by 10.7% at a cost of less than 0.3% additional die area.

Keywords: Machine Learning, Embedded Systems, Binary Neural Networks, RRAM-based Binary Dot Product Engine

1 Motivation

Machine Learning (ML) is the field of *Artificial Intelligence* (AI) that studies algorithms and statistical models aiming at teaching computer systems to perform specific tasks based on information inferred from patterns on datasets. A class of such algorithms is *Convolutional Neural Networks* (CNNs), which is applied to a vast diversity of fields, from image processing to natural language processing [1]. On the one hand, CNNs are versatile and can be used for numerous purposes, from teaching a machine how to play chess [2] to having an embedded

** At the time of this work, João Vieira was affiliated with the University of Utah.

system with a camera recognizing objects in real-time [3]. On the other hand, CNN-driven algorithms are costly both on computing power and energy demand. While having a machine or a cluster of machines learning how to play chess in a data warehouse may not be time-, compute-, or power-bounded, using an embedded system to detect objects in frames in real-time has all those constraints. Therefore, it is important to find efficient implementations of CNNs targeting embedded systems.

CNNs are composed of phases called layers. Each layer of a CNN can be seen as an independent function that takes inputs, applies a set of mathematical operations and exports an output. The output of a layer may be used as the input of another layer. Usually, each layer has a significant amount of inputs and outputs that may have to be transferred multiple times from and to the memory subsystem throughout the algorithm's execution. Also, statistics show that 90% of the mathematical operations executed in the layers of CNNs are convolutions. For these reasons, the key to achieve efficient implementations of CNN-driven algorithms relies on simultaneously reducing data movements and optimizing the convolution operation.

In the literature, two distinct approaches for optimizing the convolution operation are widely adopted. One consists of using dedicated accelerators capable of leveraging convolution's potential parallelism, and data proximity. The other exploits the adoption of lower precision operands in CNNs aiming at speeding up their execution while not degrading severely the quality of the results.

The use of dedicated accelerators aims at leveraging both the convolution's potential parallelism and the high bandwidth to the memory, which is a characteristic of accelerators. As an example, ISAAC [4] re-purposes the hardware resources of *Resistive Random-Access Memory* (RRAM) devices to enable massive parallel computation that can be used to accelerate certain layers of CNNs. ISAAC is not only capable of performing thousands of operations simultaneously but also does so in-situ, i.e., it does not require to move the operands out of the memory device. An important shortcoming of ISAAC, however, is that it hardly fits the power constraints of embedded systems since RRAM devices require a significant amount of energy to operate. Plus, in general, dedicated accelerators that do not operate at memory level like ISAAC present an overhead associated with transferring the operands from the memory hierarchy that is only surpassed by the benefits of processing large datasets. Thus, when the dataset is not big enough, the benefits of using the accelerator might not cover the cost of transferring the data to and from the accelerator. Since CNN-driven algorithms executing on embedded devices usually work with small datasets, suitable for the device's capacity, most accelerators are not suited for such systems.

In parallel, the use of lower precision operands in CNNs is also widely adopted, aiming at simplifying the operations executed by the several layers and, consequently, increasing performance. Intel's bfloat16 is a floating-point format using only 16 bits that was created with the purpose of being used by CNNs. Ultimately, the precision of the operands can be reduced to the limit of 1 bit per operand. *Binary Neural Networks* (BNNs), CNNs that use only 1 bit per

operand, show significant performance boosts over CNNs and drastically reduce the memory requirements as they require only 1 bit to store each operand. Although reducing the precision of the operands to the limit sacrifices the quality of the results to some level, studies show that XNOR-Nets, a class of BNNs, still achieve admissible quality for many applications. Due to the drastic reduction of requirements, some BNNs can be efficiently implemented in embedded systems.

An additional advantage brought by BNNs is the increase in data redundancy. Part of the operands involved in the mathematical operations in the CNN layers are intrinsic to those layers. These operands, called weights, are grouped in the form of kernels, which are often small in size (typical values are 3×3, 5×5, and 7×7). When using 32-bit integers or floating-points to represent the weights, it is unlikely to find two kernels whose weights are exactly the same. However, when using only 1 bit, chances are that some kernels will have exactly the same weights by the same order. Thus, such kernels are redundant and there is no need to have them both stored in memory.

The system proposed in this work [5] leverages all the three aspects discussed in the last paragraphs (dedicated hardware structures, lower-precision operands, and data redundancy brought by the adoption of lower-precision operands) to accelerate the execution of BNNs in embedded systems. To achieve this, we propose a novel *Functional Unit* (FU) based on the work presented on [6], the *Binary Dot Product Engine* (BDPE), to be integrated into the execution path of a general-purpose *Central Processing Unit* (CPU). The purpose of the BDPE is both to store the most used kernels, hence avoiding transferring them from the memory, and accelerate the binary convolution operation, by leveraging the low-level properties of the RRAM technology. Results show that the proposed system provides significant benefits both in terms of performance improvements and energy efficiency over the high-efficiency processor ARM Cortex-A53 at a cost of a negligible area overhead.

All in all, the main contributions of this work are:

1. We revisit the work proposed in [6] and modify their artifact to create a novel FU [5];
2. We propose the architecture of a BDPE capable of storing redundant binary kernels and executing the binary convolution operation efficiently;
3. We integrate the novel BDPE with the execution path of the ARM Cortex-A53;
4. We propose an ARMv8 *Intruction Set Architecture* (ISA) extension for supporting the functionality of the novel BDPE;
5. We present a simulation methodology using accurate models and complete frameworks to assess the benefits of the proposed system.

Throughout the following sections, we will discuss the proposed system, namely how the different components were implemented and the methods and tools that were used. The rest of this Chapter is organized as follows: Section 2 introduces the most important concepts about CNNs and BNNs as well as the process of leveraging RRAM-enabled computing to perform convolutions; Section 3 summarizes the working principles of the binary convolution block used

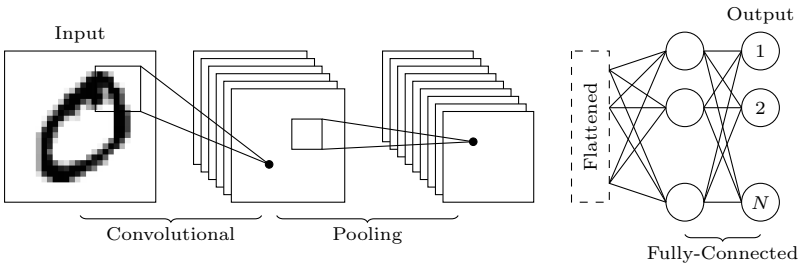


Fig. 1: CNN overview highlighting the three most common types of layers: Convolutional, Pooling, and Fully-Connected.

as base for the novel BDPE. Section 4 details the architecture of the proposed system. The evaluation methodology and the main results of this work are presented in Section 5. Section 6 presents some related work. Finally, Section 7 closes this Chapter.

2 Background

This section introduces the reader to important concepts that are used in this work. First, the basic structure and working principles of CNNs are explained, focusing on the particular kind of CNNs used in this work, BNNs. Then, the working principle of RRAM computation applied to CNNs is detailed.

2.1 Convolutional Neural Networks

CNNs constitute a class of Deep Learning algorithms that are particularly useful for Computer Vision applications. Applied to that field, a CNN assigns importance (learnable weights) to the features of images that allows differentiating different objects depicted in those frames. First, several images are presented to the CNN together with known information about them (e.g., a picture of a cat and the information that the picture contains a cat). The image is processed by the CNN and the weights are updated depending on the deviation of the obtained result from the expected result. This process goes on until certain convergence conditions are met. This phase is called training. After training a CNN, it can be used for inferring information from images that were not shown to it during the training phase. For instance, given a picture of a cat different from those shown during training, the CNN will be able to detect that there is a cat in the picture based on all the cat pictures it has seen during training. This phase is called inference.

A CNN is composed of different phases called layers. The type, quantity, and order of those layers are responsible for the properties, namely the accuracy, of the CNN. Each layer of a CNN takes inputs (in the case of the first layer, the inputs are the pixels of an image), performs a group of mathematical operations

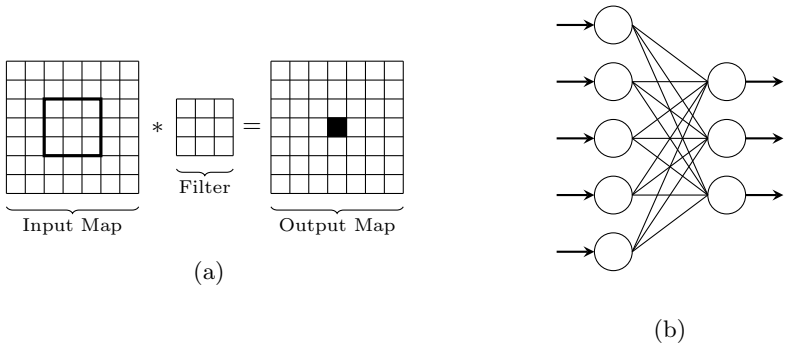


Fig. 2: (a) Convolutional layer; (b) Fully-Connected layer.

over those inputs and export outputs. The inputs of a layer are organized in the form of an input feature map, whereas the set of outputs is called output feature map. Except for the last layer, the outputs of the layers serve as inputs for the next layer. Depending on the mathematical operations realized over the input data, the layers are grouped under different categories. The three most common layer categories are convolutional, fully-connected and pooling, as shown in Fig. 1. Convolutional layers perform the convolution operation between the input and a given kernel (a kernel is an organized array of weights), as shown in Fig. 2a. In fully-connected layers, each element of the input has a weighted influence over each element of the output, as shown in Fig. 2b. Polling layers are transformation layers that perform a mathematical, possibly non-linear, operation over the elements of the input. Such layers can be used for transforming the input such as resizing or rectifying. Examples of possible transformations are max-pooling and *Rectified Linear Unit* (ReLU), respectively, which are depicted in Fig. 3. Note that two of the three most common types of layers, convolutional and fully-connected, implement mathematical operations that can be unrolled in a set of convolutions.

Both the convolutional and the fully-connected layers of CNNs use operands that are intrinsic to those layers (the kernels of the convolutional layers and the weights of the fully-connected layers). Those values are responsible for the quality of the results exported by the CNN, and are calculated during the training phase.

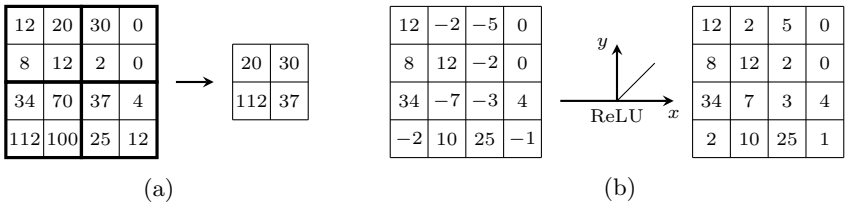


Fig. 3: Two widely used Pooling layers in CNNs: (a) Max-Pool; (b) ReLU.

At the beginning of the operation, all the weights are initialized to random values. Then, a set of images and corresponding information are fed to the CNN. The CNN processes those inputs, and based on the deviation between the calculated results and the expected, updates the weights using feedback processes.

Depending on several factors such as the quality of the training set, the training configuration and duration and the architecture of the CNN itself, the trained CNN is able to detect more or less accurately objects in new frames that were not shown during the training phase. A popular metric to assess the quality of the inference is the *mean Average Precision* (mAP). The mAP is defined as the average value of another metric called *Average Precision* (AP), which is a composed metric that takes into account several factors, such as precision, recall, and *Intersection over Union* (IoU). Precision measures the accuracy of the CNN's predictions, i.e., the percentage of correct predictions; recall is the percentage of positives that the CNN is able to detect, regardless of the assigned class; and IoU measures the overlap between the real area where an object is located in a frame and the area where it is detected by the CNN. mAP values are included in the interval $[0, 1]$, being 1 optimal. The mAP can also be used to compare CNNs, which is particularly useful when assessing the loss of quality due to applying heuristics for increasing execution performance (e.g., when reducing the precision of the operands).

2.2 XNOR-Based Binary Convolution

BNNs are a particular class of CNNs where both the inputs of the layers and the weights are specified by only one bit and their values are limited to -1 and 1. Duo to this representation, the atomic operations of the regular convolution can be replaced by simpler operations. The element-wise multiplication is replaced by the XNOR operation whereas the accumulation is replaced by bit-count [7]. Binarizing the operands and replacing the costly operators in the convolution by simpler ones significantly increases the performance of the convolution operation. Since convolutions constitute approximately 90% of the total operations implemented by CNNs [8], optimizing this operation significantly impacts the overall performance and energy efficiency associated with executing the CNN.

Furthermore, binarizing a CNN also reduces significantly the memory requirements duo to each operand being represented by a single bit. For instance, when comparing with a 32-bit representation of the operands and the weights, the memory requirements are reduced $32\times$. However, reducing the precision of the operands to the limit of 1 bit comes at the cost of sacrificing accuracy. While CNNs such AlexNet suffer from accuracy losses lower than 10% when binarized, some other networks may be rendered totally useless. Therefore, not all CNNs are viable for binarization.

2.3 YOLO: You Look Only Once

Similarly to AlexNet, YOLO [3] is another example of a CNN that can be binarized without sacrificing severely its accuracy. YOLO is a state-of-art CNN for

real-time object detection targeting low-end devices. It divides the image into regions and predicts bounding boxes and associated probabilities for each region. Unlike expensive object-detection-oriented CNNs such as R-CNN [9] and Fast R-CNN [10], which require thousands of network evaluations to detect multiple objects, YOLO looks at the whole image at testing time, so its predictions are informed by global context. As a result, the YOLO performs 1000× faster than R-CNN and 100× faster than Fast R-CNN [3]. By sizing the network, YOLO also allows trading accuracy for performance by reducing the number of network layers. This represents an advantage for embedded devices characterized by low computing power. When binarized using the definitions in [7], the XNOR-Net version of a given configuration of YOLO, YOLOV3-tiny XNOR-Net, shows approximately the same mAP as the full-precision network, while reducing the size of the weights approximately 32× and thus executing up to 58× faster. For the *Street View House Numbers* (SVHN) dataset [11], the YOLOV3-tiny XNOR-Net shows a mAP decrease of only 0.43%, while increasing the number of detections by 75%.

Duo to the binary nature of both the inputs and the operands in binary convolutions, which allows replacing the expensive multiply and accumulate operations by simpler XNOR and bit-counts, BNNs have great potential to be further accelerated using dedicated structures in hardware. One of the most popular approaches to do so is to re-purpose the hardware resources of RRAMs to enable computation.

2.4 RRAM-Powered Convolution Acceleration

RRAM bit-cells, also called memristors, are devices with two terminals composed by metal electrodes and a switching oxide stack [12]. The application of a certain programming voltage between the two electrodes changes the conductivity of the metal oxide, which leads to a switch between two stable resistance states: *Low-Resistance State* (LRS) and *High-Resistance State* (HRS). Thus, allowing to encode the binary values 1 and 0, respectively. The switching from HRS to LRS is called a set process and can be achieved through the application of a positive programming voltage. The complementary process is denominated a reset process and involves the use of a negative programming voltage.

Although the classic applications of memristors only use two possible states to encode binary data, the physical characteristics of memristors allow the LRS to assume a wide range of values. This particular feature allows to use RRAM cells to store non-binary values (encoded as physical impedance), which can be used to implement hardware analog dot-products. To do that, the RRAM cells are organized into a crossbar of passive arrays, removing all the access transistors. The output current becomes the sum of the currents flowing through each memristor, which has encoded a certain non-binary value in the form of impedance, and thus, it becomes the sum of the input voltages weighted by the values encoded in the memristors, as shown in Fig. 4a. Duo to the high density of RRAMs, in theory, using RRAM-based analog computation enables massive parallelism leading to remarkable performance and energy efficiency

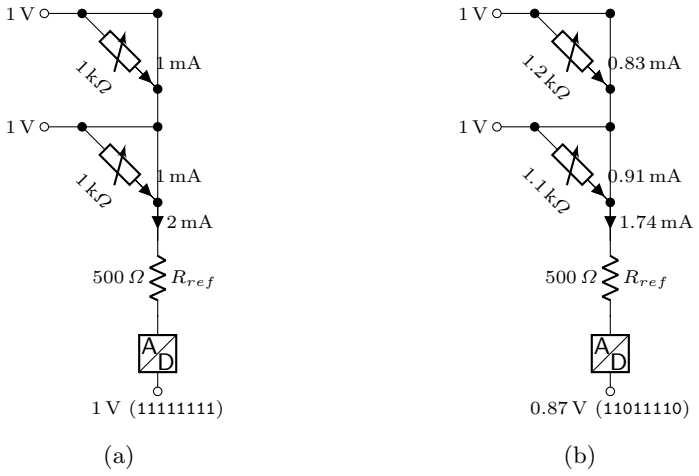


Fig. 4: Operation principle of the RRAM-powered analog dot product: (a) under perfect conditions; (b) in the presence of variations. When subjected to variations, the final sum of the currents flowing to the memristors has an associated error which is reflected in the output of the 8-bit ADC.

benefits. However, RRAM-based analog computation suffers from two limitations that largely reduce its applicability. First, it requires a massive quantity of *Digital-Analog Converters* (DACs) and *Analog-Digital Converters* (ADCs) that are area- and power-hungry. This limits the scalability of RRAM-based analog compute devices. Second, memristors are subjected to variability, which has its source in process variations during fabrication as well as temperature oscillations during the device operation. This severely decreases the usability of the analog capabilities of RRAMs, as the results are severely tampered by the lack of precision of the resistance values. The effect of varying the resistance values of the memristors is depicted in Fig. 4b.

The work presented in [6], which also uses RRAM arrays to perform computation, mitigates this effect by allowing only two levels of resistance and performing the dot product in a purely digital way, without relying on an analog sum of currents. Unlike analog-based RRAM computation, the approach used in that work allows obtaining reliable results that minimize the possibility of errors. Thus, the convolutional block presented in [6] served as inspiration for this work and was further developed to become a fully-functional FU and integrated within the pipeline of a general-purpose CPU.

3 RRAM-Based Binary Convolution Block

The use of dedicated accelerators to increase performance when executing CNN-based applications is one of the most well-succeeded approaches. Dedicated accelerators take the most advantage of the hardware resources for a very specific

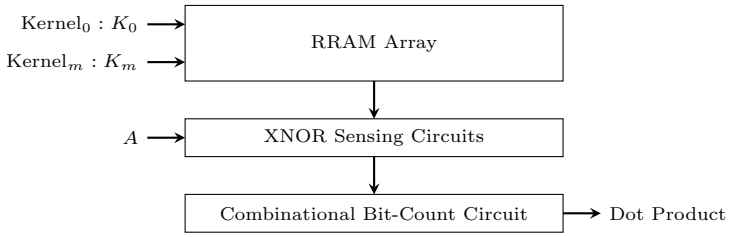


Fig. 5: Structure of the RRAM-based binary convolution block proposed in [6].

task, enabling massive performance boosts. Furthermore, they usually have their own high-bandwidth channels to memory. However, such accelerators are expensive in terms of hardware and power demand, rendering them unsuitable for embedded devices, which are highly hardware- and power-constrained. Therefore, our system uses a custom BDPE that is integrated within the processor execution path. Our unit is based in the binary convolution engine presented [6] that efficiently implements the binary convolution operation.

Duo to the nature of the operands in a binary convolution (each element of the operands has only 1 bit), the costly operations of multiplication and accumulation required by the regular convolution can be replaced by XNOR and bit-count, respectively [7], which can be implemented efficiently in hardware using the low-level properties of memristors and some additional circuitry. Note that in the context of CNNs, convolutions are implemented as a sequence of several dot products. Therefore, the block that implements the binary convolution executes a sequence of dot products, one per cycle, until producing the final result of the convolution.

Fig. 5 shows the top-level structure of the convolutional block proposed in [6]. The sequences of binary weights are stored in the RRAM array, one per row. To perform one dot product with an input vector, the line of the RRAM containing the binary weights is selected and the input data is applied to the RRAM array. The bit-wise XNOR of the input and the binary weights stored in the selected line is performed as a readout of the RRAM array. Then, the result of the XNOR operation is applied to the input of the bit-count circuit, which counts the number of bits set to `'1'`. Finally, the bit-count circuit exports the result of the dot product.

Structure-wise, the part of the convolution block responsible for implementing the XNOR operation is composed of three parts: the precharge circuit; the programming circuit; and the XNOR circuit, as shown in Fig. 6. The precharge circuit loads the outputs (*out* and $\overline{out}$) to V_{DD} before the reading. The programming circuit serves to store the kernels in the RRAM array prior to computation. The XNOR circuit is responsible for performing the XNOR operation between the input and the kernel stored in the selected row of the RRAM.

Overall, the operation of the convolutional block is divided into two phases: (1) the kernel storage phase; and (2) the computing phase.

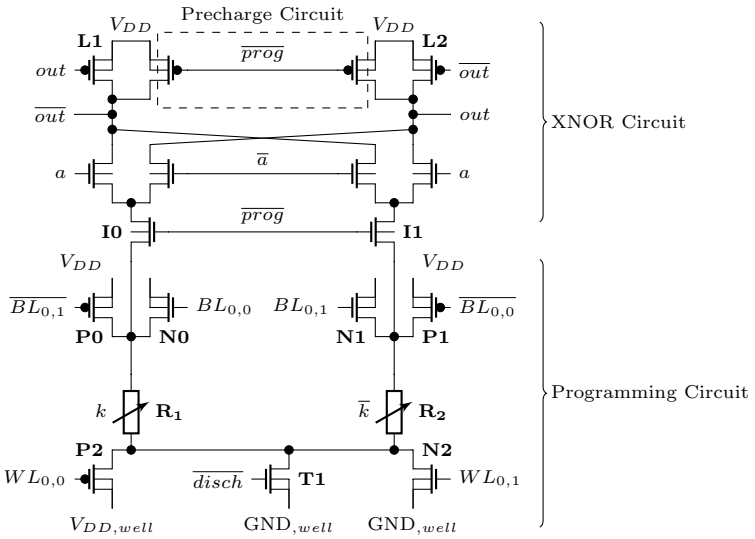


Fig. 6: 1-bit RRAM-based XNOR cell that serves as unit of the RRAM array used in the convolution block. All terminals illustrated as disconnected are connected to GND.

During the programming phase ($prog = 1$), both the outputs, out and $\overline{out}$, are precharged to V_{DD} and the RRAM array is isolated from the XNOR circuit through transistors $I0$ and $I1$. Additionally, $\overline{disch}$ is set to GND, turning $T1$ off. The value of k is stored in the RRAM in a complementary way. For example, when $k = 0$, R_1 is set to HRS and R_2 to LRS. R_1 is set to HRS by turning on transistors $N0$ and $P2$ and turning off transistors $P0$, $P1$, $N1$ and $N2$. During the reset process of R_1, $V_{DD,well}$ and $GND_{,well}$ are switched to $2 \times V_{DD}$ and V_{DD}, respectively. As such, the voltage difference across R_1 becomes $-V_{prog} = -2 \times V_{DD}$, which triggers a reset process. Similarly, R_2 is set to LRS by turning on transistors $P1$ and $N2$ and turning off transistors $P0$, $P2$, $N0$ and $N1$. Considering $k = 1$, R_1 is set to LRS and R_2 to HRS in a similar way.

The computing phase consists of calculating the XNOR between an input a and a weight k stored in the memory, which is performed as a single memory readout thanks to the XNOR sense amplifiers. The read sequence goes as follows: first, $V_{DD,well}$ and $GND_{,well}$ are switched to V_{DD} and GND respectively. Signals $prog$ and $\overline{disch}$ are set to `'0'` and `'1'`, respectively. Hence, nodes out and $\overline{out}$ are grounded through the RRAMs and transistor $T1$. During this phase, the complementary resistances of the RRAMs control the discharge currents, so that the memristor in LRS is discharged faster than the one in HRS. Using a deep N-well also eliminates any breakdown risk or reliability degradation on the transistors [13]. When out or $\overline{out}$ reach the voltage $V_{DD} - V_{T,pmos}$ ($V_{T,pmos}$ denotes the threshold voltage of transistors $L1$ and $L2$), the associated $pmos$ transistor is

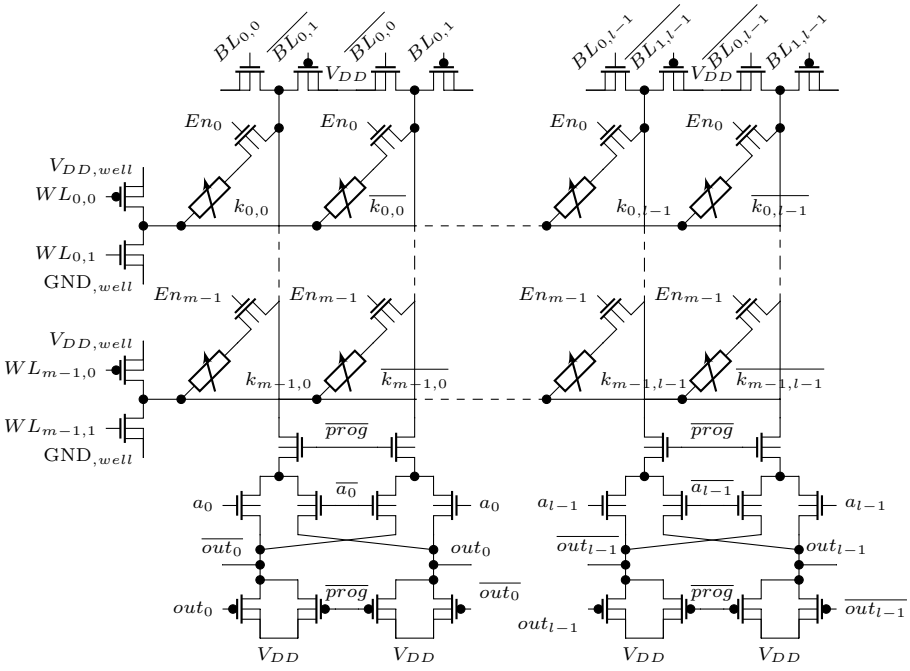


Fig. 7: RRAM-based XNOR $m \times l$ array organization. All terminals illustrated as disconnected are connected to GND.

turned on, pulling up the other node (out or $\overline{out}$) to V_{DD}. The resulting output is the XNOR of a and k.

Considering the operation principles above described, it is possible to organize several XNOR cells into a matrix. Figure 7 shows a $m \times l$ RRAM matrix array, where m is the number of kernels to be stored and l is the number of weights of each kernel. For example, storing ten 3×3 kernels requires a 10×9 RRAM array. From the proposed bit-cell structure of Fig. 6, all the XNOR part is shared between all the structures in each pair of columns and is put at the bottom acting as a digital sense amplifier. Similarly, the programming circuits are shared at each column and row to be able to individually address all the RRAM lines. Sharing the programming transistors also allows parallel programming.

As shown in Fig. 8, a convolution between an input feature map of size $n \times n$ and a kernel (also $n \times n$, for simplicity) leads to an output feature map with n^2 elements. The kernel is unrolled into a vector so the n^2 weights are encoded in the n^2 pairs of RRAM cells. In this example, n^2 is the number of columns of the RRAM array illustrated in Fig. 7 ($N^2 = l$). Element $o_{i,i}$, the central element of the output feature map, is calculated by performing the dot product of the unrolled input feature map ($A = \{a_{0,0}, \ldots, a_{n-1,n-1}\}$) with the unrolled kernel

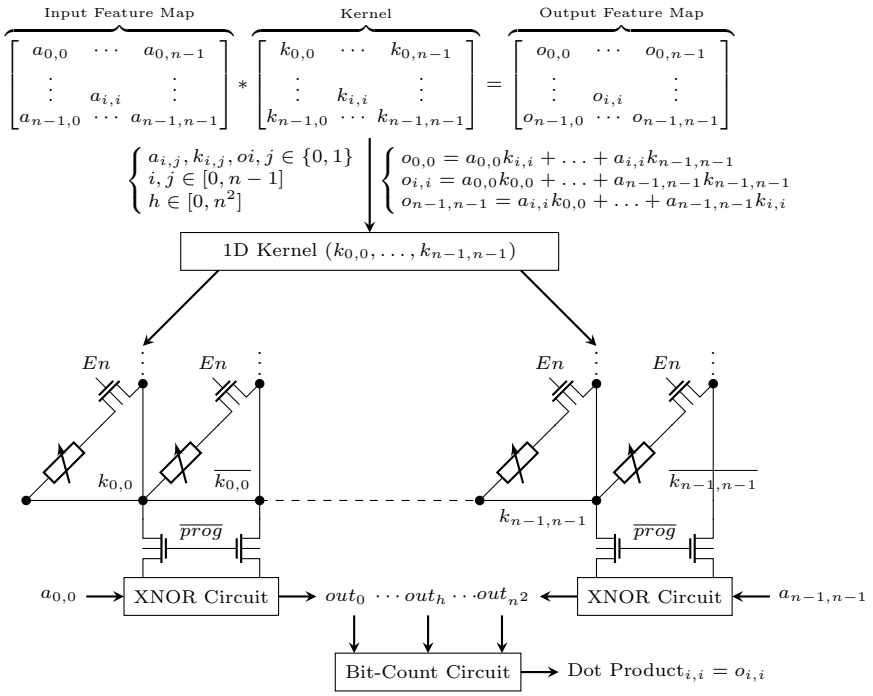


Fig. 8: Example of binary dot product between two $n \times n$ matrices. All terminals illustrated as disconnected are connected to GND.

($K = \{k_{0,0}, \ldots, k_{n-1,n-1}\}$) by selecting the row of the RRAM where the kernel is stored through signal En_x and summing the partial results, $out_h (h \in [0, n^2])$, through an external circuit to perform the bit-count operation.

Results reported by [6] show that the binary convolutional block surpasses previous analog RRAM-based accelerators in terms of reliability against RRAM and CMOS process variations. Furthermore, it shows significant energy efficiency increase when compared both with conventional *Multiply-Accumulate* (MAC) solutions and state-of-the-art accelerators such as ISAAC [4].

4 CPU Integration

To be able to take advantage of the convolutional block presented on [6] and fully integrate it within the execution path of a general-purpose CPU as a regular FU, some adaptations to the original architecture of the convolution block were required, as explained in [5].

The advantages of using the convolutional block are two: (1) performing the XNOR operation efficiently using the RRAM array; (2) calculating the bit-count atomically. To use the RRAM array to calculate the XNOR between an input and a vector of binary weights it is required the vector of binary weights

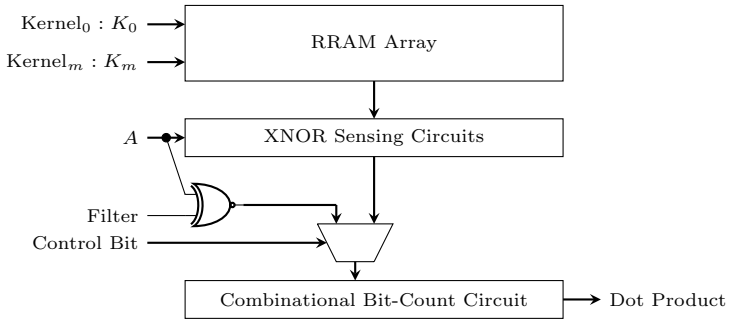


Fig. 9: Block diagram of the proposed BDPE.

to be stored in the RRAM array. However, having all the weights stored in the RRAM array is not feasible as it would result in a tremendous chip area overhead. Therefore, only a small number of weights can actually be stored inside the BDPE. Nevertheless, the bit-count circuit does not depend on the weights being stored in the RRAM array. Therefore, to avoid rendering useless the entire convolutional block whenever the weights are not stored in the RRAM array, additional logic was added that allows performing the XNOR operation between two binary input vectors using classic CMOS logic gates. This way, the benefit of performing the bit-count using the circuit of the original block can still be leveraged. Figure 9 illustrates the architecture of the novel BDPE highlighting the modifications to the original convolution block.

The integration of the BDPE with a general-purpose CPU is divided into three phases: (1) the integration with the processor's pipeline; (2) the creation of new instructions in the CPU ISA to use the unit; and (3) provide compiler support to use the added ISA instructions in the software side. The CPU choice for integrating the BDPE was the ARMv8-A ARM Cortex-A53 due to both being a high-efficiency low-end processor that provides a competitive baseline and also the available simulations models for this particular CPU.

As shown in Fig. 10, the BDPE is integrated with the processor's pipeline in the *Execute* stage, similarly to the *Arithmetic and Logic Unit* (ALU), and stores the results of the computation in the *Execute/Memory Access* pipeline register. According to the ARM Architecture Reference Manual for the ARMv8-A architecture profile [14], the ARMv8 ISA has unused *opcodes* that can be re-purposed to expand the functionality of the CPU. Using two of the unused *opcodes*, two instructions were created and assigned to the novel BDPE.

Fig. 11 illustrates the format of the new instructions and denotes the purpose of each distinct set of bits. Each of the new instructions is decoded in the *Decode* stage such that the content of the register specified by `rm` serves as input data of the BDPE; the content of the register represented by `rn` is the input kernel; `imm6` specifies the address of the kernel stored in the RRAM array; and the second *Most Significant Bit* (MSB) of the *opcode* designates the control bit.

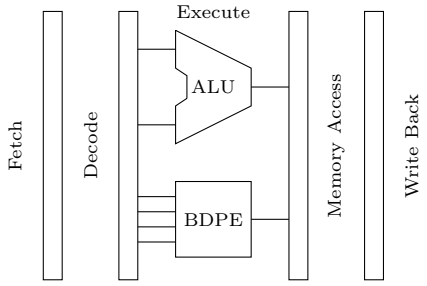


Fig. 10: Simplified block diagram of a generic processor pipeline integrating the proposed BDPE.

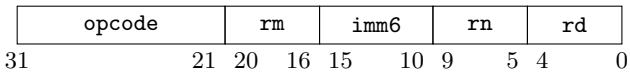


Fig. 11: Format of the new instructions added to the ARMv8 ISA to allow the processor to issue instructions to the BDPE. The *opcodes* `10000011000` and `11000011000` were re-purposed to specify the custom instructions. `rm` and `rn` specify addresses of 64-bit registers; `imm6` represents a 6-bit immediate; and `rd` specifies the address of the destination 64-bit register.

As shown in Fig. 12, the workflow for running a BNN using the novel BDPE is divided into profiling and execution. During the profiling, the BNN is used to perform a single inference while the kernel space is profiled, selecting the most frequently used kernels. The selected kernels are stored in the RRAM, and a configuration file is generated containing the information about the content of the RRAM. Then, the CNN is recompiled, and the code responsible for implementing the binary convolution is replaced by custom code that utilizes the BDPE. If the kernel being used is stored in the RRAM, the compiler inserts a special instruction to perform the binary convolution using the RRAM array. Otherwise, the compiler inserts a load instruction to fetch the kernel from memory, followed by a special instruction that performs the binary convolution using the two data inputs of the BDPE.

5 Results and Discussion

To assess the performance and energy efficiency improvements enabled by the BDPE, a light configuration of the YOLOV3 XNOR-Net was used, the YOLOV3-tiny. First, the network was trained using Darknet [15], a C-based CNN framework on a general-purpose computer for the dataset SVHN [11]. Then, the network was profiled to obtain statistics about the usage of the kernels during the inference phase. Due to the number of kernels used by the CNN, storing all of them inside the RRAM array would lead to a prohibitively large circuit. Thus, only the kernels that are used the most are stored inside the RRAM array while

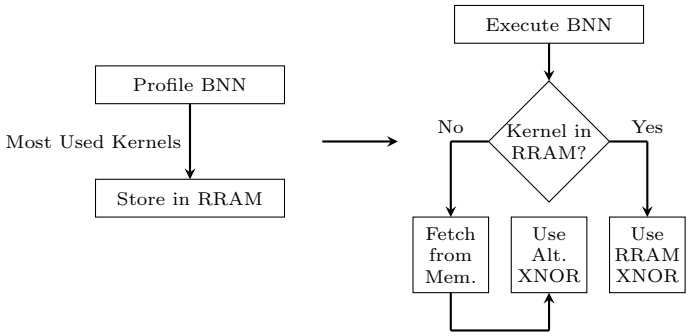
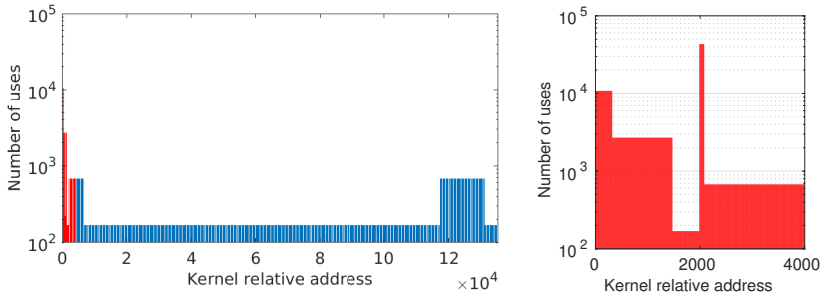


Fig. 12: Simple example that illustrates the process of storing the most used kernels inside the RRAM and running a BNN using the novel BDPE.

the remaining kernels have to be transferred from memory during inference. As result of profiling YOLOV3-tiny XNOR-Net, it was concluded that a small portion of the kernels is used in a significant percentage of the convolution operations, as suggested by Fig. 13. More specifically, 0.07% of the kernels are used in 9.74% of the convolutions. Therefore, those kernels are stored in the RRAM array.

To further evaluate the impact of storing different amounts of kernels inside the BDPE, five scenarios were considered where the RRAM usage rate (percentage of convolutions that use kernels locally stored in the RRAM) varies between 10% and 50% when executing YOLOV3 XNOR-Net.



(a) (b)

Fig. 13: Number of times each kernel is used during YOLOV3-tiny XNOR-Net inference: (a) entire kernel space; (b) zoom in showing a spike on the usage of a very limited set of kernels.

5.1 Performance Analysis

A modified version of the gem5 architectural simulator [16] was used to assess the proposed system performance benefits. Due to gem5's known inaccuracies [17], the default ARM model was improved with the gem5-X framework [18], allowing to reduce the ARM model's error margin from 10% to 4% for the ARM Cortex-A53. The system was simulated in *System Emulation* (SE) mode by compiling all the inputs of the network into a single executable binary file. Additionally, the Darknet framework was modified at assembly level to use the custom BDPE instead of the processor's ALU when performing binary convolutions.

To determine the most used kernels and populate the RRAM array, the following two-step procedure was used: (1) Darknet was ran using gem5 and the kernel space was profiled; (2) The most used kernels were selected and stored in the RRAM. After populating the RRAM, the gem5 module responsible for emulating the BDPE was rebuilt. Because the framework was not recompiled, gem5 in SE mode mapped the data structures to the same addresses used in (1), and the application flow was kept the same except for the binary convolutions involving the most frequently used kernels stored in the RRAMs. In those cases, the RRAM array was used instead of the alternative XNOR mechanism to perform the XNOR operation. The complete system featuring the modified ARM Cortex-A53 and four DRAM ranks of 1 GB each operating at 2400 MHz was emulated and the entire workflow of Darknet was executed.

As result of offloading the execution of binary convolutions to the BDPE, the kernels that are stored in the RRAM array are not requested from memory during inference. Thus, a reduction in memory accesses equal to the RRAM usage rate is observed, as shown in Fig. 14a. Additionally, it can be observed that over 99% of the memory accesses reduction happens at the L1 cache, suggesting that the system has the maximum benefits of caching effects.

However, avoiding the transfer of sequential kernels to the processor whenever the RRAM array is used produces irregularities in the memory access patterns. This situation leads to more evictions and cache collisions, thus causing additional cache misses, as shown in Fig. 14b. Nevertheless, the increase of the cache misses is lower than 0.01% relative to the total number of memory accesses. Hence, this side-effect is negligible and does not affect the overall performance.

All in all, as illustrated in Fig. 14c, for a usage rate of 10% the performance improvement is 11.3%. Also, the performance gains show no significant variation with the RRAM usage rate. This effect has two main causes: (1) both the data paths in the BDPE take exactly one cycle to perform a binary convolution; (2) due to caching effects, the kernels are stored in the L1 cache 94% of the time, substantially reducing the time required to fetch them. Consequently, using the alternative method for performing the XNOR of the kernel and the input data takes approximately the same time as using the RRAM array and does not impact negatively the overall performance.

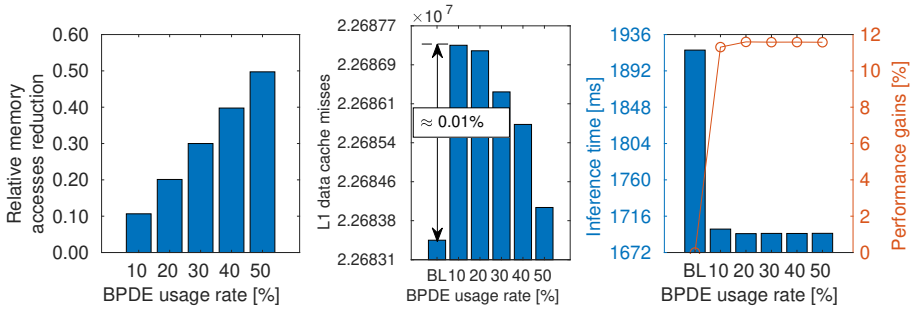


Fig. 14: (a) Number of memory accesses spared depending on the RRAM usage rate when compared with the baseline system; (b) Number of L1 data cache misses caused by BDPE-driven memory access irregularities; (c) Inference time and performance gains of the baseline system and the proposed system considering five RRAM usage rates.

5.2 Hardware Resources

Circuit-level metrics were obtained through electrical simulations using a commercial 28 nm *Fully Depleted Silicon On Insulator* (FD-SOI) design kit to assess the hardware requirements and the power demand of the devised BDPE. Delay and power results were extracted from Eldo simulations, to be used in models for the architectural evaluation. These metrics were extracted for the two possible cases to consider when the XNOR is performed using a kernel locally stored in the RRAM or a kernel coming from the processor's registers, respectively. In order to consider an average case, it was assumed that half of the data inputs, as well as the kernels, are zeros and the other half is ones. For the area estimation, the full-custom layout of the RRAM array and its associated control path were modeled using Cadence Virtuoso. The bit-count circuit was synthesized with Synopsys Design Compiler from *Register Transfer Level* (RTL) netlists and integrated into a Place & Route flow using Cadence Innovus to obtain the complete layout of a 256×64 RRAM-based BDPE.

Table 1 shows the hardware requirements, power demand and delay for the BDPE. In practice, since a 10% RRAM usage rate allows achieving the best trade-off between hardware requirements, performance improvements and energy savings, that scenario was used to obtain the results in this section.

The die area required to implement the novel mechanism is only $3.845\,\mu m^2$ per CPU core, using a FD-SOI 28 nm process, while a dual-core ARM Cortex-A53 in an equivalent process requires $2.8\,mm^2$ [19]. Therefore, the BDPE represents less than 0.3% of the total CPU area. The energy spent for a single operation when using the RRAM array (*control bit=0*) is reduced by 37% comparing to using the alternative mechanism (*control bit=1*). This is allowed by the intrinsic energy efficiency of the RRAM array [6]. Although this advantage comes

Table 1: Hardware resources and average power demand of the BDPE considering the two possible paths data paths considering a RRAM usage rate of 10%. When *control bit=0*, the RRAM array is used to implement the XNOR operation. Otherwise, the alternative XNOR mechanism is used.

	Area/Hardware resources [μm^2]	**Power** [mW]	**Delay** [ps]
control bit=0	3,845	1.24	408
control bit=1		3.23	214

at a cost of a delay overhead at circuit level, the maximum operating frequency allowed is still 2.5 GHz. Thus, as the target platform is the ARM Cortex-A53 with an operating frequency of 2 GHz, the BDPE can be integrated with the system without constraining its overall frequency.

5.3 Energy Efficiency

As a secondary result of running the modified version of Darknet, gem5 produced timing results, statistics on memory accesses and usage of the CPU's several modules. Such results were applied to 28 nm FD-SOI power models for ARMv8 in-order cores, proposed by [18] and [20], allowing to estimate energy consumption.

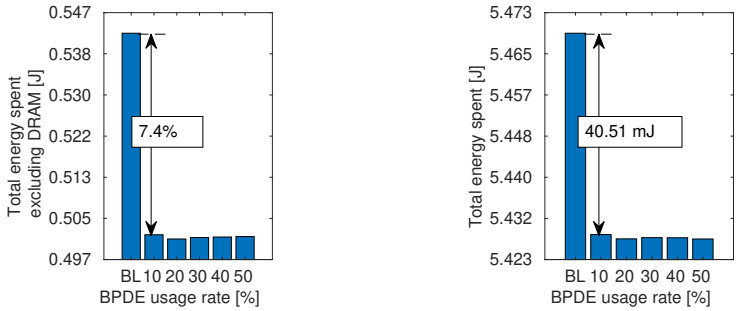
The total energy spent by the baseline system (ARM Cortex-A53) and the five scenarios using the BDPE is illustrated in Fig. 15a. Then, Fig. 15b shows the energy consumption for the same circumstances subtracted by the energy spent by the DRAM. As shown in Table 2, the total energy spent by the BDPE is negligible when compared with the rest of the system, and so the energy savings are mostly due to the reduction in the execution time. As the execution time is approximately constant regardless of the RRAM usage rate, so are the energy savings. When considering only the processing system (excluding the DRAM main memory), the use of the BDPE allows for average energy savings of 7.4%.

6 Related Work

As the need for executing compute- and power-hungry CNNs in hardware- and power-constrained devices arises, novel approaches to execute these algorithms efficiently are proposed. Accelerators such those presented in [21–24] aim at reducing data movements by taking advantage of data redundancy, which results

Table 2: Total energy spent by the BDPE and the CPU during the inference phase of YoloV3 XNOR-Net.

RRAM usage rate [%]	**Baseline**	**10**	**20**	**30**	**40**	**50**
BDPE [µJ]	0	0.870	0.279	0.271	0.263	0.255
CPU [µJ$\times$**10^6**]	0.542	0.502	0.501	0.501	0.501	0.502



(a) (b)

Fig. 15: Energy spent by the baseline and the proposed system during the inference phase of YOLOV3-tiny XNOR-Net: (a) excluding the energy spent by the DRAM; (b) including the energy spent by the DRAM.

in significantly increasing the energy efficiency. Other approaches such as that presented in [25] consist of a method to tolerate errors resulting from aggressive memory voltage scale down, which also allows increasing the energy efficiency significantly.

The use of the compute capabilities of RRAM devices is also an approach to further improve the energy efficiency of CNNs that has shown great interest in the past few years. Shafiee *et al.* [4] proposed using the memristors of the RRAM arrays to store the weights in the form of impedance and perform dot products in an analog fashion. However, the impedance precision of memristors is limited by the ADCs and DACs used to interface the analog RRAM array as well as variations that affect the resistive devices both during its fabrication, operation, and lifetime. Furthermore, ADCs and DAC consume a lot of energy and occupy a lot of chip area, which limits the circuit scalability.

Recent works such as [26, 27] proposed using RRAMs to implement dot products in a binary using *Current Sense Amplifiers* (CSAs) or reduced precision ADCs. These approaches use only two logical levels per memristor, which not only improves energy efficiency but also increases the robustness of the devices when compared with RRAM-based analog computation. However, these solutions failed to study important design issues such as the offset voltage of the *Sense Amplifiers* (SAs), which may lead to operational failures [28].

Xiaoyu *et al.* [29] proposed a parallel XNOR-RRAM array using CSAs. Their work shows that by carefully partitioning the RRAM-array, the SA offset issue was alleviated and the proposed architecture is robust against CMOS and RRAM process variations. However, the assumed RRAM resistance variation was very optimistic (4.5% whereas some work reported around 20% of resistance variation [30]), and the operation of the proposed architecture still may fail under realistic assumptions. Moreover, most of the recent works either did not investigate the impact of RRAM process variations at the circuit level [27, 4, 31–33] or mainly

focused on the architectural level, lacking of proper circuit level evaluations [26, 27, 4].

Furthermore, previous work still fails to provide embedded-systems-oriented devices that are capable of operating under tight hardware and power constraints. Our work aims at filling these gaps. It has its base on a fully digital RRAM convolutional block that mitigates the limitations of previous RRAM-based solutions, such as errors due to RRAM process variations. Plus, our engine is tiny compared to previous solutions, presenting a chip overhead of merely 0.3% over the CPU used for assessment. Nevertheless, it allows performance improvements up to 11.3% and energy savings as high as 7.4%. Another important contrast between previous solutions and our proposal is the methodology for using the device. While previous approaches rely on non-standard interfaces and protocols for programming and sending data (when such methodology is addressed at all), our system provides a full-stack implementation of a FU that can be easily addressed using assembly language.

7 Conclusions

Our work consists of a novel RRAM-based BDPE suited for accelerating the inference phase of BNNs and meant to be integrated within the pipeline of a general-purpose CPU. To our knowledge, this is the first attempt at exploring the acceleration of CNNs through custom RRAM-based CPU-integrated FUs, which makes it an important contribution. The power demand, hardware resources and propagation delay of the devised mechanism were modeled, and its impact on the considered base system was comprehensively evaluated using the Darknet framework and gem5. Results showed that the novel BDPE achieved performance improvements of 11.3% and 7.4% energy savings. Furthermore, the integration of the novel mechanism requires only few modifications to the baseline CPU, while representing less than 0.3% of the total die area, and does not lower the operation frequency of the system.

The reported advantages allowed by the devised system are tightly coupled with the considered baseline CPU and the used CNN model. Since this work uses an ARM Cortex-A53, which is a high-efficiency CPU, the compute power and energy efficiency enabled by the baseline puts it among the most efficient embedded systems. Nevertheless, the use of the devised BDPE still allows achieving significant performance improvements and energy savings at the cost of a minor area overhead. It is also worth saying that should the baseline be a more rudimentary processing system (e.g., an ultra-low-power embedded system), and the novel BDPE would allow for bigger improvements. Furthermore, the RRAM usage rate is highly affected by the CNN choice. While YoloV3-tiny XNOR-Net allowed for a RRAM usage rate of 10%, other CNNs might allow for higher usage rates. While this may not be relevant performance-wise, as explored in Section 5, higher RRAM usage rates lead to fewer data movements and also to performing the XNOR operation using the RRAM array instead of the alternative CMOS mechanism, which impacts positively the energy efficiency.

Acknowledgements

This work was primarily supported by the grant 2016016 from the United States-Israel Binational Science Foundation.

Other supporting grants are SFRH/BD/144047/2019 from Fundação para a Ciência e a Tecnologia (FCT), Portugal; ERC Consolidator Grant COMPUSAPIEN (GA No. 725657); ERC starting grant Real-PIM-System (GA No. 757259); and EC H2020 EUROLAB4HPC2 project (GA No. 800962).

References

1. Collobert, R., Weston, J.: A unified architecture for natural language processing: deep neural networks with multitask learning. In: ICML. Volume 307 of ACM International Conference Proceeding Series., ACM (2008) 160–167
2. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419) (2018) 1140–1144
3. Redmon, J., Divvala, S.K., Girshick, R.B., Farhadi, A.: You only look once: Unified, real-time object detection. In: CVPR, IEEE Computer Society (2016) 779–788
4. Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J.P., Hu, M., Williams, R.S., Srikumar, V.: ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In: ISCA, IEEE Computer Society (2016) 14–26
5. Vieira, J., Giacomin, E., Qureshi, Y.M., Zapater, M., Tang, X., Kvatinsky, S., Atienza, D., Gaillardon, P.: A product engine for energy-efficient execution of binary neural networks using resistive memories. In: VLSI-SoC, IEEE (2019) 160–165
6. Giacomin, E., Greenberg-Toledo, T., Kvatinsky, S., Gaillardon, P.: A robust digital rram-based convolutional block for low-power image processing and learning applications. IEEE Trans. on Circuits and Systems **66-I**(2) (2019) 643–654
7. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: ECCV (4). Volume 9908 of Lecture Notes in Computer Science., Springer (2016) 525–542
8. Cong, J., Xiao, B.: Minimizing computation in convolutional neural networks. In: ICANN. Volume 8681 of Lecture Notes in Computer Science., Springer (2014) 281–290
9. Girshick, R.B., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: CVPR, IEEE Computer Society (2014) 580–587
10. Girshick, R.B.: Fast R-CNN. In: ICCV, IEEE Computer Society (2015) 1440–1448
11. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning. In: NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011. (2011)
12. Wong, H.P., Lee, H., Yu, S., Chen, Y., Wu, Y., Chen, P., Lee, B., Chen, F.T., Tsai, M.: Metal-oxide RRAM. Proceedings of the IEEE **100**(6) (2012) 1951–1970
13. Tang, X., Giacomin, E., Micheli, G.D., Gaillardon, P.: Circuit designs of high-performance and low-power rram-based multiplexers based on 4t(ransistor)1r(ram) programming structure. IEEE Trans. on Circuits and Systems **64-I**(5) (2017) 1173–1186

14. ARM: Arm architecture reference manual (2018)
15. Redmon, J.: Darknet: Open source neural networks in c. http://pjreddie.com/darknet/ (2013–2016)
16. Binkert, N.L., Beckmann, B.M., Black, G., Reinhardt, S.K., Saidi, A.G., Basu, A., Hestness, J., Hower, D., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Altaf, M.S.B., Vaish, N., Hill, M.D., Wood, D.A.: The gem5 simulator. SIGARCH Computer Architecture News **39**(2) (2011) 1–7
17. Butko, A., Garibotti, R., Ost, L., Sassatelli, G.: Accuracy evaluation of GEM5 simulator system. In: ReCoSoC, IEEE (2012) 1–7
18. Qureshi, Y.M., Simon, W.A., Zapater, M., Atienza, D., Olcoz, K.: Gem5-x: A gem5-based system level simulation framework to optimize many-core platforms. In: SpringSim, IEEE (2019) 1–12
19. Abouzeid, F., Bernicot, C., Clerc, S., Daveau, J., Gasiot, G., Noblet, D., Soussan, D., Roche, P.: 30% static power improvement on ARM cortex®-a53 using static biasing-anticipation. In: ESSCIRC, IEEE (2016) 37–40
20. Pahlevan, A., Qureshi, Y.M., Zapater, M., Bartolini, A., Rossi, D., Benini, L., Atienza, D.: Energy proportionality in near-threshold computing servers and cloud data centers: Consolidating or not? In: DATE, IEEE (2018) 147–152
21. Du, L., Du, Y., Li, Y., Su, J., Kuan, Y., Liu, C., Chang, M.F.: A reconfigurable streaming deep convolutional neural network accelerator for internet of things. IEEE Trans. on Circuits and Systems **65-I**(1) (2018) 198–208
22. Chen, Y., Krishna, T., Emer, J.S., Sze, V.: Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. J. Solid-State Circuits **52**(1) (2017) 127–138
23. Jo, J., Kim, S., Park, I.: Energy-efficient convolution architecture based on rescheduled dataflow. IEEE Trans. on Circuits and Systems **65-I**(12) (2018) 4196–4207
24. Sim, J., Park, J., Kim, M., Bae, D., Choi, Y., Kim, L.: 14.6 A 1.42tops/w deep convolutional neural network recognition processor for intelligent ioe systems. In: ISSCC, IEEE (2016) 264–265
25. Kim, S., Howe, P., Moreau, T., Alaghi, A., Ceze, L., Sathe, V.S.: Energy-efficient neural network acceleration in the presence of bit-level memory errors. IEEE Trans. on Circuits and Systems **65-I**(12) (2018) 4285–4298
26. Ni, L., Liu, Z., Yu, H., Joshi, R.V.: An energy-efficient digital reram-crossbar-based cnn with bitwise parallelism. IEEE Journal on Exploratory solid-state computational devices and circuits **3** (2017) 37–46
27. Tang, T., Xia, L., Li, B., Wang, Y., Yang, H.: Binary convolutional neural network on RRAM. In: ASP-DAC, IEEE (2017) 782–787
28. Agbo, I., Taouil, M., Hamdioui, S., Weckx, P., Cosemans, S., Raghavan, P., Catthoor, F., Dehaene, W.: Quantification of sense amplifier offset voltage degradation due to zero-and run-time variability. In: ISVLSI, IEEE Computer Society (2016) 725–730
29. Sun, X., Yin, S., Peng, X., Liu, R., Seo, J., Yu, S.: XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks. In: DATE, IEEE (2018) 1423–1428
30. Chen, A., Lin, M.R.: Variability of resistive switching memories and its impact on crossbar array performance. In: 2011 International Reliability Physics Symposium, IEEE (2011) MY–7
31. Xia, L., Tang, T., Huangfu, W., Cheng, M., Yin, X., Li, B., Wang, Y., Yang, H.: Switched by input: power efficient structure for rram-based convolutional neural network. In: DAC, ACM (2016) 125:1–125:6

32. Chen, X., Jiang, J., Zhu, J., Tsui, C.: A high-throughput and energy-efficient rram-based convolutional neural network using data encoding and dynamic quantization. In: ASP-DAC, IEEE (2018) 123–128
33. Chi, P., Li, S., Xu, C., Zhang, T., Zhao, J., Liu, Y., Wang, Y., Xie, Y.: PRIME: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In: ISCA, IEEE Computer Society (2016) 27–39